In Proceedings of the IJCAI-03 Workshop on Mixed-Initiative Intelligent Systems, Acapulco, Mexico, August 2003.

Mixed-initiative Control for Teaching and Learning in Disciple

Mihai Boicu[1], Gheorghe Tecuci[1,2], Dorin Marcu[1], Cristina Boicu[1], Bogdan Stanescu[1]
[1] Learning Agents Laboratory, George Mason University
MSN 4A5, 4400 University Dr., Fairfax, VA, 22030, USA
[2] Center for Strategic Leadership, US Army War College
Carlisle Barracks, PA 17013, USA
{mboicu, tecuci, dmarcu, ccascava, bstanesc}@gmu.edu

Abstract

Disciple is an approach to agent development by subject matter experts where an expert teaches a Disciple agent his or her problem solving expertise in a way that resembles how a person teaches another person. This paper presents an overview of the teaching and learning process during which the expert helps Disciple to learn and Disciple helps the expert to teach it, emphasizing the mixed-initiative control of the component agents of Disciple, particularly the Modeling agent, the Rule Learning agent and the Exception-based Ontology Learning agent. It discusses current implementations, evaluation results, lessons learned and future developments.

1 Introduction

Disciple is an evolving theory, methodology, and family of agent shells for the development of intelligent agents by subject matter experts, with limited assistance from computer scientists or knowledge engineers [Tecuci *et al*., 2001]. A subject matter expert interacts directly with a Disciple learning agent, to teach it to solve problems, in a way that is similar to how the expert would teach a human apprentice, by giving the agent examples and explanations, as well as by supervising and correcting its behavior.

Building the knowledge base of a Disciple agent is an example of a problem that, by its very nature, requires a mixed-initiative solution. Indeed, neither the subject matter expert, nor the Disciple agent can solve this problem independently. While the subject matter expert has the knowledge to be represented in the knowledge base, he is not a knowledge engineer and cannot properly formalize it. On the other hand, the learning agent obviously does not have the knowledge to be represented, but it can incorporate knowledge engineering methods to formalize the expert's knowledge. The goal is then to divide the responsibility between the expert and Disciple for those elements of knowledge engineering for which they have the most knowledge and aptitude, such that together they form a complete team for knowledge base development. This requires mixed-initiative reasoning, where the expert and the agent share representations, communicate naturally, properly divide their tasks and responsibilities, coordinate their actions, take initiative and release control.

We have pursued the research on the Disciple approach for many years, developing increasingly more competent learning agents [Tecuci, 1988, 1998; Boicu, 2002]. Mixed-initiative, however, is a newer and very promising feature of the Disciple approach, which is discussed in this paper.

The next section provides a brief overview of the teaching and learning process in Disciple. Section 3 describes the multi-agent architecture of the Disciple Teaching agent and the mixed-initiative control of its component agents which include the Modeling agent, the Rule Leaner and the Exception Handler. Then, sections 4, 5, and 6 present more details on the mixed-initiative control inside these component agents. Finally, section 7 concludes the paper with a discussion of related research on mixed-initiative reasoning.

2 Teaching the Disciple agent

A Disciple agent solves problems through task reduction. That is, it successively reduces a problem solving task to simpler tasks, finds the solutions of the simplest tasks, and then successively composes these solutions to obtain the solution of the initial task. To exhibit this type of behavior, its knowledge base has to contain an object ontology (that specifies the terms from a particular domain) and reduction and composition rules (expressed with the terms from the ontology).

A fragment of the object ontology from the Center of Gravity analysis domain [Tecuci *et al*., 2002] is represented in the bottom left hand side of Figure 1. The ontology is a hierarchical representation of the objects and types of objects from the application domain. It represents the different kinds of objects, the properties of each object, and the relationships existing between objects.

The expert teaches Disciple how to solve problems by considering a certain problem solving task, helping the agent to understand each reasoning step toward the solution, and supervising and correcting the agent's behavior, when it attempts to solve new problems. During mixed-initiative interactions the agent learns from the expert, building and refining its knowledge base to represent the problem solving expertise of the human expert.

First the expert expresses his/her reasoning process in natural language, as illustrated by the task reduction example in the upper left side of Figure 1. The top task is the task to be reduced. In order to reduce this task the expert asks a relevant question and formulates its answer. The answer to this question leads to the reduction of this task to a subtask. Disciple may help the expert in completing the current problem solving step by employing different heuristics, such as analogical reasoning with previously encountered problem solving steps.

Once the reduction step is defined, the expert has to help the agent to understand it. The agent has to identify those elements from the object ontology that represent the meaning of the question-answer pair. These ontology elements are the explanation of why the top task is reduced to the bottom task. One explanation piece is:

President_Roosevelt is_protected_by US_Secret_Service_1943

Once these explanation pieces are found, the agent generalizes the task reduction example and its explanation to the task reduction rule from the right hand side of Figure 1. The learned rule has an informal structure (shown in the top right part of Figure 1) and a formal structure (shown in the bottom right part of Figure 1). The informal structure preserves the natural language of the expert and is used in agent-user communication. The formal structure is used in the actual reasoning of the agent. The reduction rule is an IF-THEN structure that expresses how and under what conditions a certain type of task may be reduced to simpler subtask. Notice, however, that the formal structure of the rule does not have a single applicability condition, but a plausible upper bound condition and a plausible lower bound condition. These conditions approximate the exact condition that Disciple is attempting to learn. The agent will use this partially learned rule in problem solving, and the feedback received from the expert will be used to learn the exact condition. Notice that the generalization of the example into a rule is based on the object ontology which is used as the agent's generalization hierarchy. Indeed, the specific instances from the example (e.g. President_Roosevelt, US_Secret_Service_1943) are replaced in the learned rule with more general concepts from the object ontology (i.e. head_of_government, personal_protection_agency), and their relationships (e.g. is_protected_by, provides).

3 The architecture of the Teaching Assistant

Figure 2 shows the hierarchical multi-agent architecture of a major Disciple component, the Teaching Assistant. The parent agents call the child agents, each agent running in a separate process. The top level process in Dis-

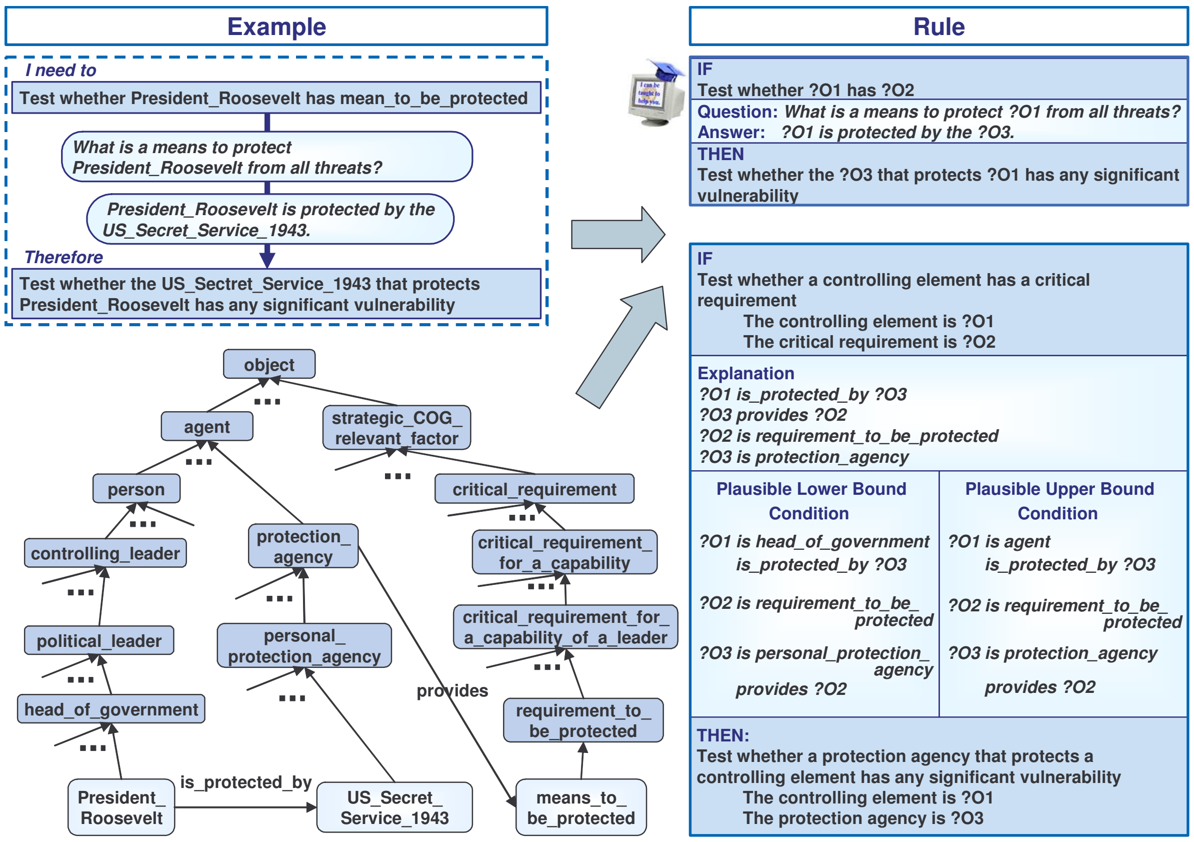
Example
I need to
Test whether President_Roosevelt has mean_to_be_protected
What is a means to protect President_Roosevelt from all threats?
President_Roosevelt is protected by the US_Secret_Service_1943.
Therefore
Test whether the US_Sectret_Service_1943 that protects President_Roosevelt has any significant vulnerability
object
agent
strategic_COG_relevant_factor
person
critical_requirement
controlling_leader
protection_agency
critical_requirement_for_a_capability
political_leader
personal_protection_agency
critical_requirement_for_a_capability_of_a_leader
head_of_government
provides
requirement_to_be_protected
President_Roosevelt
is_protected_by
US_Secret_Service_1943
means_to_be_protected
Rule
IF
Test whether ?O1 has ?O2
Question: What is a means to protect ?O1 from all threats?
Answer: ?O1 is protected by the ?O3.
THEN
Test whether the ?O3 that protects ?O1 has any significant vulnerability
IF
Test whether a controlling element has a critical requirement
The controlling element is ?O1
The critical requirement is ?O2
Explanation
?O1 is_protected_by ?O3
?O3 provides ?O2
?O2 is requirement_to_be_protected
?O3 is protection_agency
Plausible Lower Bound Condition
?O1 is head_of_government
is_protected_by ?O3
?O2 is requirement_to_be_protected
?O3 is personal_protection_agency
provides ?O2
Plausible Upper Bound Condition
?O1 is agent
is_protected_by ?O3
?O2 is requirement_to_be_protected
?O3 is protection_agency
provides ?O2
THEN:
Test whether a protection agency that protects a controlling element has any significant vulnerability
The controlling element is ?O1
The protection agency is ?O3

Figure 1: Teaching the Disciple agent

ciple is the Mixed-Initiative Control Agent that controls the interaction between the user and Disciple. The user may initiate this process by specifying the problem solving task to be performed. Then Disciple attempts to successively reduce the current task to simpler subtasks.

When Disciple does not know how to reduce the current task (such as the top task from the left hand side of Figure 1), the Control Agent invokes the Modeling Assistant to help the user to provide a solution. The Modeling Assistant invokes the Example Editor, the Example Completion Agent, and the Example Analyzer. These agents interact with the user, suggesting plausible completions of the example and checking its correctness. For instance, as the user specifies the question (see Figure 1), the Example Editor updates the current internal structure of the example. It also invokes the Word Completion Agent that suggests plausible completions of the word that is being typed by the user. When the question is completed, the Example Completion Agent suggests a plausible answer of the question and even a plausible subtask. When the entire example is completed, the Example Analyzer may suggest improvements or corrections. The mixed-initiative interactions between these agents will be discussed in more detail in section 4.

After the example from the left hand side of Figure 1 is specified, the Task Formalization Assistant helps the user to formalize the new tasks which will be included in the formal structure of the rule to be learned (shown in the bottom right hand side of Figure 1). Then the example is passed to the Rule Learning Agent. Rule learning is a complex mixed-initiative process in which the user helps the agent to understand why the example is correct, and the agent help the user to explain it. There are several agents involved in this process. The Explanation Generation Agent proposes plausible explanations to the user, based on user hints, analogical reasoning, and natural language processing. The Implicit Explanations Agent identifies contextually true explanations and automatically adds them to the explanations of the current example. The Rule Analyzer checks the learned rule and identifies potential problems. This agent initiates an internal problem solving process to check if the rule generates too many solutions, in which case it provides hints for additional explanations of the example constraining the rule.

The Rule Refinement Assistant is also composed of several agents that interact with the user to improve the rule based on its successful or unsuccessful use in problem solving. Similarly, the Exception-Based Ontology Learning Assistant interacts with the user to identify extensions of the object ontology that eliminate the exceptions of the learned rules.

Because the interface of Disciple is implemented in Java, the Graphical User Interfaces (GUI) of all the agents run in the same process. However, each agent controls the life and interaction of its graphical components, making the GUI control a distributed process shared by all the component agents of Disciple.

These types of interaction have led to a synergistic integration of the main teaching processes: modeling, learning and problem solving [Boicu, 2002].

An important lesson learned from this architecture is related to the model used to implement the mixed-initiative control. We have initially implemented a centralized rule-based control that is simple, uniform, and a single place in the system for self-awareness. However, we discovered that it has two major disadvantages. First, all the mixed-initiative processes need to be based on the same communication scheme which is not suitable for all the agents. Second, all the communications need to go through the central mixed-initiative control, which creates bottlenecks when there are processes that need to exchange many messages and the multi-tasking platform

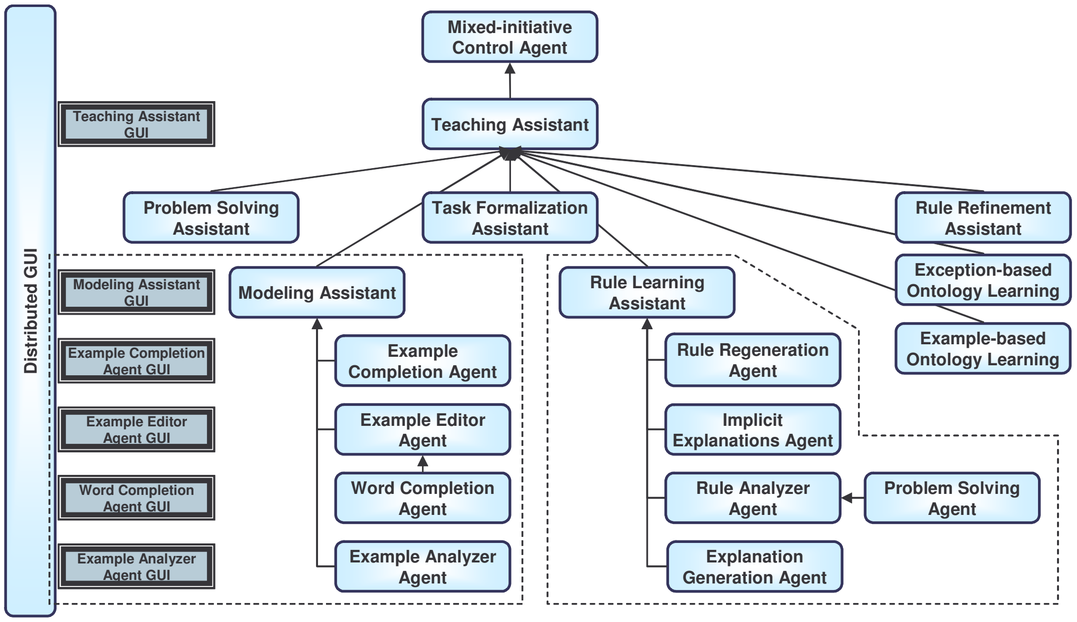
Mixed-initiative Control Agent
Teaching Assistant
Teaching Assistant GUI
Problem Solving Assistant
Task Formalization Assistant
Rule Refinement Assistant
Distributed GUI
Modeling Assistant GUI
Modeling Assistant
Rule Learning Assistant
Exception-based Ontology Learning
Example-based Ontology Learning
Example Completion Agent GUI
Example Completion Agent
Rule Regeneration Agent
Example Editor Agent GUI
Example Editor Agent
Implicit Explanations Agent
Word Completion Agent GUI
Word Completion Agent
Rule Analyzer Agent
Problem Solving Agent
Example Analyzer Agent GUI
Example Analyzer Agent
Explanation Generation Agent

Figure 2: The multi-agent architecture of the Teaching Assistant

is slow. Therefore we are currently developing an alternative approach to the mixed-initiative control. This new approach is an integration of hierarchical control and mixed-initiative control. There is a top-level mixed-initiative control that involves the user and the top level assistants from Disciple which perform a major function identifiable by the user, such as problem solving, modeling, task formalization, rule learning, etc. However, each such assistant (e.g. the Modeling Assistant) may initiate a new mixed-initiative process that involves its subagents (i.e. the Example Editor, the Example Completion Agent, and the Example Analyzer). If one of the subagents has also a multi-agent architecture, then it may initiate a mixed-initiative process for its subagents, and so on.

The highest level of mixed-initiative control is governed by the Mixed-initiative Control Agent. This agent, as a parent of the other agents, may request and receive detailed reports on the status of all the others agents (through its direct sub-agents). Therefore, at this level it is still possible to perform global self-awareness reasoning as needed, but without a permanent burden on the component agents. This agent may also control the mixed-initiative interaction between Disciple and external agents.

Figure 3 describes the other levels of mixed-initiative control in Disciple. Agent-k is initiated by its parent agent. The parent agent controls its life cycle, being responsible of the correct startup and shutdown of Agent-k. At startup the parent agent registers the new agent with all the existing running agents that need to know about it. Then Agent-k executes its initialization task which may involve initiation of sub-agents and/or communication with existing agents. After initialization Agent-k enters an execution cycle, performing tasks corresponding to the external messages received, as it will be discussed below. When a shutdown request is received Agent-k is responsible of the proper shutdown of its sub-agents and the corresponding cleanup procedure.

Agent-k communicates with other agents (including its subagents) using two types of messages: task execution requests and result notifications. There might be two or more agents capable to execute the requested task. The negotiation is coordinated by the sender agent, resulting in one of the following situations: 1) all capable agents execute the task and the results are unified; 2) the best agent is selected to execute the task; and 3) the first capable agent available executes the task.

During or after the execution of a task an agent may send notification messages to the registered agents, informing them about the results obtained during task execution. The receiving agents may start the execution of new tasks or modify the current internal execution flow.

Notice that the same agent may run in different processes (for instance the problem solving agent is invoked both at the main level of the Teaching Assistant but also to help the Rule Analyzer).

The current implementation pre-codes this mixed-initiative mechanism. Moreover, the agents are responsible for implementing critical communication and control capabilities. We are currently working on developing a mixed-initiative framework that will offer predefined types of mixed-initiative agents, which will make the definition of such agents very easy, reducing them to the description of tasks and their execution.

4 The Modeling Assistant

In this section we will present the mixed-initiative control at the level of a Disciple assistant, using the Modeling Assistant as an example.

Let us consider the top left task from Figure 1 which Disciple does not know how to reduce. In such a case the user has to provide the reduction from the left hand side of Figure 1 (i.e. the question, the answer and the subtask), and the Modeling Assistant has to help the user to specify it.

There are two main situations in the interaction between the user and the Modeling Assistant: 1) when the user is typing a word of the current knowledge element (e.g. a word of the question) and 2) when the user has just finished the current element and wants to continue with the next one (e.g. has finished typing the question, and wants to start typing the answer). We will discuss each of them.

Figure 4 shows the exchanged messages and the actions taken by each agent of the Modeling Assistant when the user is typing a word of the question in the Example Editor. The elementary user action is the typing of a letter (for instance, the letter "r" in: "What is a means to protect Pr"). The GUI of the Example Editor is notifying all the registered agents of the user action:

User modified question (example-id, new-question, version)

As a result, the Example Editor does the following:

- identifies concepts and instances in the current Question fragment (none in this case);
- modifies the internal structure of the example;

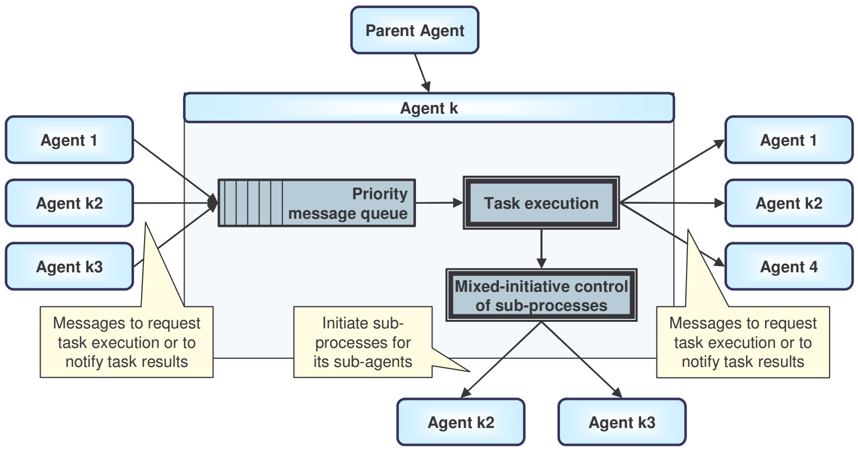
Parent Agent
Agent k
Agent 1
Agent k2
Agent k3
Priority message queue
Task execution
Mixed-initiative control of sub-processes
Agent 1
Agent k2
Agent 4
Messages to request task execution or to notify task results
Initiate sub-processes for its sub-agents
Messages to request task execution or to notify task results
Agent k2
Agent k3

Figure 3: Mixed-initiative control architecture

- if any concept or instance was identified, notifies all the registered processes on the actions taken: "Updated question (example-id, new-question, version)"

There are two agents to notify: the Example Editor GUI and the Word Completion Agent. The Example Editor GUI will need to synchronize the current form (which may be different if the user typed other letters) with the received form of the question (which will contain newly identified concepts or instances). The agents use the "version" parameter for synchronization. Each time the user is performing an editing operation the version of the edited element is increased. When an update notification is sent with a result the Example Editor GUI compares the received version with its current one, and, if it is the same, it updates the result. Otherwise it ignores the received result.

A similar process takes place for suggesting the word completions. When the Word Completion GUI receives the update message it invalidates the current selection list, keeps the version and waits for the new list of suggestions. The Word Completion Agent will compute the most plausible completions, based on partial lexical matching, reasoning context analysis and analogical reasoning. If, during computation, another message is received, the current computation will be discarded and a new one will start. After the list is computed the agent sends the completions to the GUI. The synchronization is again assured through the use of the version parameter.

Another important aspect of the control mechanism is the proactive execution. For instance, the Example Completion agent is running in the background to hypothesize the most plausible answers (or the patterns of the answers) of the currently edited question. Each time the question is modified this agent updates its hypotheses. When the user finishes the question by pressing Enter, this agent immediately proposes the most plausible answers. The agent uses several heuristics methods to generate plausible answers. They are based on natural language processing, analogy with answers from previously learned rules, and methodological guidelines, as described in Boicu [2002].

An earlier version of the Modeling Assistant was evaluated in an experiment at the US Army War College in Spring 2002. In this experiment, which will be repeated in Spring 2003, 15 subject matter experts have taught personal copies of Disciple their own reasoning in Center of Gravity analysis. The evaluation results described in Boicu [2002] show that this mixed-initiative implementation has led to significantly better results than a previously used example editor. The experts succeeded to model more examples, and needed less help from the knowledge engineers, because the Modeling Assistant provided suggestions that they considered generally useful and understandable. They also considered this assistant generally well organized and easy to use.

5 The Rule Learning Assistant

In this section we present the mixed-initiative control at the level of the Rule Learning Assistant, as we are currently implementing.

Let us consider again the task reduction step from Figure 1. The user and the sub-agents of the Rule Learning Agent, shown in Figure 2, have to cooperate to find the following formal explanation pieces:

President_Roosevelt is_protected_by US_Secret_Service_1943
US_Secret_Service_1943 is protection_agency
protection_agency provides means_to_be_protected
means_to_be_protected is requirement_to_be_protected

First, the Rule Regeneration Agent takes control and generates the initial version of the rule. This initial version is generated based only on the task reduction step.

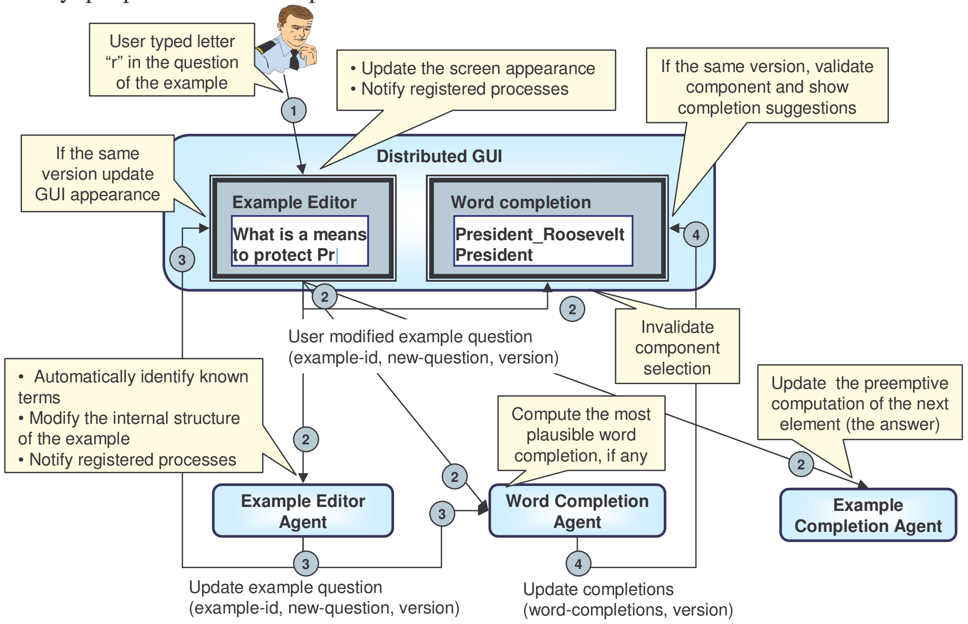
User typed letter "r" in the question of the example
1
• Update the screen appearance
• Notify registered processes
If the same version, validate component and show completion suggestions
Distributed GUI
If the same version update GUI appearance
Example Editor
What is a means to protect Pr
Word completion
President_Roosevelt
President
3
4
2
2
User modified example question (example-id, new-question, version)
Invalidate component selection
• Automatically identify known terms
• Modify the internal structure of the example
• Notify registered processes
Update the preemptive computation of the next element (the answer)
Compute the most plausible word completion, if any
2
2
2
Example Editor Agent
3
Word Completion Agent
Example Completion Agent
3
4
Update example question (example-id, new-question, version)
Update completions (word-completions, version)

Figure 4: Illustration of the mixed-initiative execution

Each time an explanation piece is accepted or deleted, the Rule Regeneration Agent recomputes the rule to incorporate or remove a generalization of the explanation piece.

Next, the Implicit Explanations Agent takes control and searches for implicit explanations. Some explanations are implicit in certain terms. For instance, "US_1943 has_as_government goverment_of_US_1943" is implicit in the name government_of_US_1943. Others are explanation pieces from upper level task reduction steps that are considered as implicit in the current reduction step (e.g. Allied_Forces_1945 has_as_member US_1945). However, in the Disciple methodology we view each example as generating a stand-alone rule. This makes the rules independent from their reasoning context, and allows their use independently of other rules. But, this requires each and every rule to contain all the necessary explanations in its applicability condition. Therefore, the Implicit Explanations Agent analyzes the rule, the example and the problem solving context and determines the most plausible implicit explanations. These explanations are automatically accepted and integrated into the rule by the Rule Explanations Agent, but the user may delete any of them.

Next, the Rule Analyzer, the Explanation Generation Agent, and the user engage in a mixed-initiative process where each of them contributes to the explanation of the example according to its capabilities.

The Rule Analyzer continuously analyzes the rule after each modification, to determine if it is good enough for problem solving. The Rule Analyzer performs both an internal analysis of the rule and also calls the Problem Solving Agent to externally check the rule. The internal check is based on the variable instantiation flow, our modeling methodology and several knowledge engineering guidelines. The external check analyzes how the rule will apply during problem solving, for instance if it generates too many solutions for a given input. The Rule Analyzer combines the results of the performed checks and displays a list of plausible problems and their suggested solutions as soon as they are found. This process keeps the user informed and assures that a better rule is learned from the very beginning. This method is applied both during rule learning and during rule refinement based on additional positive and negative examples. Experimental evaluation results showed that the users consider the suggestions provided by the agent generally useful and understandable [Boicu, 2002].

The Explanation Generation Agent has two main tasks: to propose an initial set of plausible explanations, and to help the user through mixed-initiative hint refinement. To identify the initial set of explanation pieces, the agent uses analogical reasoning and other heuristics.

At any time the user may interrupt the computation of the Explanation Generation Agent by specifying a hint that narrows down the search. A hint is a fragment of an explanation, such as an object or a relationship between two objects. Then the agent starts looking for explanations conforming to the hint. If the search process is too time-consuming or there are too many found explanations, the agent proposes only incomplete fragments of explanations (e.g. means_to_be_protected is ...) and then continues to search in the background. The user can select such an incomplete explanation fragment and give it as a hint to further narrow down the search. When the user accepts an explanation piece as correct, the Rule Regeneration Agent recomputes the rule and the Rule Analyzer updates its problems and suggestions list.

In the current implementation the Explanation Generation Agent does not have any sub-agents. We are currently making a new design for this agent that will include a sub-agent for each type of heuristic and hint-based search method. The Explanation Generation Agent will be able to start such sub-agents and integrate the explanations generated by each of them, ordered by their plausibility.

6 Exception-based ontology learning

We are currently implementing a mixed-initiative behavior for the Exception-based Ontology Learning Assistant. Because the object ontology is generally incomplete, the learned rules will accumulate exceptions. We have developed methods that elicit from the user the missing ontological knowledge in order to remove the exceptions of the rules [Boicu *et al.*, 2003].

A planned mixed-initiative extension is to allow this assistant to start by itself when the rules have accumulated too many exceptions. Each time an exception is encountered, the Exception-based Ontology Learning Assistant will start in the background to determine whether it is appropriate to propose to the user an exception handling session. This assistant will use several heuristics to determine if such a session is necessary. One heuristic is based on the number of the accumulated exceptions of the rules. Another heuristic is based on the discovery of ontology extension candidates that have high potential to reduce the exceptions. This analysis will allow the agent to decide when to take the initiative to start the exception handling session.

Moreover, the Exception Handling module is itself redesigning to support mixed-initiative between its component agents, which will benefit the entire process.

7 Related research and conclusion

A growing number of interactive intelligent systems designed to collaborate with their users in performing various tasks exhibit mixed-initiative behavior in order to improve the effectiveness of their collaboration. [Cohen *et. al.*, 1998] analyzes a large number of theories of mixed-initiative and the corresponding implemented systems in terms of their definition for initiative, and the mechanisms for detecting the initiative taker and a switch in initiative, and proposes a classification of them in four main theories. The Disciple approach to mixed-initiative can be classified in the theory that defines initiative as "the exercising of the power or ability of a dialogue participant to suggest (or perform) a plan (or task) which is instrumental to the solving of the problem at hand". Indeed, the mixed initiative approach in Disciple is task-oriented. Moreover, in Disciple, the delegation of a task by one participant to the other is not considered a change in initiative (for example, the Explanation Generation Agent proposes plausible explanations to the user only after being explicitly asked to do so, and its invocation does not signal a change in initiative taking). Finally, proposing a task to be addressed does not guarantee that the participant which proposed it will maintain the initiative until a new task or subtask is proposed (for example, the user may select a specific reasoning step for modeling, but the Example Completion Agent may very well take the initiative by proposing a question for that reasoning step).

In designing the mixed-initiative interactions between Disciple and its user, and between the internal agents of Disciple, we followed an approach similar to the one described in [Hartrum and DeLoach, 1999]. We incrementally developed both our mixed-initiative approach and Disciple, by iterating through domain-level design, agent-level design, component design and system design. Disciple is a multi-agent system that contains several task-specific agents that collaborate according to the peer-to-peer model, in which each participant can be both a contributor to a problem solving process and a receiver of the contribution(s) of some other agent(s). The same model of collaboration also characterizes the mixed-initiative interactions between Disciple and its user.

Disciple has many of the characteristics of a true mixed-initiative intelligent system, because the initiative taking is not predefined at design time but determined based on the capabilities of the participants in the interaction, given the current context. For example, the Example Completion Agent may have enough relevant information to be able to propose a plausible question for the considered reasoning step, in which case it will take the initiative an propose it. If not enough information is available, the agent will make no proposal at all, and the user has to take the initiative in defining the question. The situation is similar in the case of Rule Learning Agent, where the Implicit Explanations Agent may be capable to propose the needed explanations. But, in the case it does not have enough information, the user has to take the initiative and select the needed ones.

However, Disciple currently lacks an explicit interaction model and a user model corresponding to its mixed-initiative approach. The capabilities of each agent are pre-coded, and each agent knows exactly which other agent(s) to interact with for performing each given task. This will especially affect the flexibility, the adaptability, and the ability of these agents in the various roles they could play during the interaction between them and with the user.

COLLAGEN ([Eisenstein and Rich, 2002]) is a system for building collaborative interface agents that are based on explicit formal task models that describe and control the interactions between users and the applications for which the interface agents were built. An important difference between the Disciple approach and the COLLAGEN approach is that the COLLAGEN agents operate as bridges between users and external applications, while the Disciple agents are the actual applications, and therefore the task (interaction) models are tightly integrated with the agents the experts use, having access to the full power of the agent's representation framework, its reasoning and learning capabilities.

[Fleming and Cohen, 2001] presents a user-specific quantitative framework for determining the utility of interaction (which may not necessarily imply a change in initiative). Their focus is on when it is useful for the system, while performing an action, to ask the user clarification-type questions. [Horvitz, 1999] proposes a similar framework in which the focus is on when should the system interrupt the user (which performs the action) with notifications and/or initiative taking. In contrast, Disciple currently uses a predefined fixed strategy for determining when to attempt to take the initiative and how to notify the user about it. For example, the Word Completion Agent always takes the initiative in proposing to the user a set of possible terms for insertion into the edited text and immediately updates the display once the word to the left of the keyboard cursor changes. In contrast, both the Example Analyzer and the Rule Analyzer wait until the user signals the completion of the modeling and (respectively) the learning tasks to notify the user of potential problems with the corresponding processes even though they continuously check and update their assessment of the user activities (thus allowing the user to concentrate on the creative aspect of those processes).

A future version of Disciple will include a user model that will permit a flexible assessment of the utility of interaction between the user and the agent, for all types of interactions between them, and considering the initiative change as an additional factor in the interaction.

Acknowledgements

This research was sponsored by DARPA, AFRL, AFMC, USAF, under agreement number F30602-00-2-0546, by the AFOSR under grant no. F49620-00-1-0072, and by the US Army War College.

References

[Boicu *et al.*, 2003] Cristina Boicu, Gheorghe Tecuci, Mihai Boicu, and Dorin Marcu. Improving the Representation Space through Exception-Based Learning. To appear in *Proceedings of the Sixteenth International Flairs Conference*. 2003.

[Boicu, 2002] Mihai Boicu. Modeling and Learning with Incomplete Knowledge. *PhD Thesis in Information Technology*, Learning Agents Laboratory, School of Information Technology and Engineering, George Mason University, 2002.

[Cohen *et al.*, 1998] Robin Cohen, Coralee Allaby, Christian Cumbaa, Mark Fitzgerald, Kinson Ho, Bowen Hui, Celine Latulipe, Fletcher Lu, Nancy Moussa, David Pooley, Alex

Qian, and Saheem Siddiqi. What is Initiative? *User Modeling and User-Adapted Interaction*, 8, 171–214, Kluwer Academic Publishers, 1998.

[Eisenstein and Rich, 2002] Jacob Eisenstein, and Charles Rich. Agents and GUI's from Task Models. In *Proceedings of the 7th international conference on intelligent user interfaces*, 47–54, 2002.

[Fleming and Cohen, 2001] Michael Fleming, and Robin Cohen. A User Modeling Approach to Determining System Initiative in Mixed-Initiative AI Systems. In *Proceedings of the Eighth International Conference on User Modeling*, 54–63, 2001.

[Hartrum and DeLoach, 1999] Thomas C. Hartrum, and Scott A. DeLoach.. Design Issues for Mixed-Initiative Agent Systems. *AAAI-99 Workshop on Mixed-Initiative Intelligence*, Orlando, Florida, 1999.

[Horvitz, 1999] Eric Horvitz. Principles of Mixed-Initiative User Interfaces. In *Proceedings of CHI'99, ACM SIGCHI Conference on Human Factors in Computing Systems*, Pittsburg, Pennsylvania, 159–166. ACM Press, 1999.

[Tecuci, 1988] Gheorghe Tecuci. DISCIPLE: A Theory, Methodology and System for Learning Expert Knowledge. *Thése de Docteur en Science*, University of Paris-South, France, 1988.

[Tecuci, 1998] Gheorghe Tecuci. *Building Intelligent Agents: An Apprenticeship Multistrategy Learning Theory, Methodology, Tool and Case Studies*. Academic Press, London, 1998.

[Tecuci *et al*., 2001] Gheorghe Tecuci, Mihai Boicu, Michael Bowman, and Dorin Marcu, with a preface by Murry Burke. An Innovative Application from the DARPA Knowledge Bases Programs: Rapid Development of a High Performance Knowledge Base for Course of Action Critiquing. *AI Magazine*, 22(2), 43–61. AAAI Press, Menlo Park, California, 2001.

[Tecuci *et al.*, 2002] Gheorghe Tecuci, Mihai Boicu, Dorin Marcu, Bogdan Stanescu, Cristina Boicu, and Jerome Comello. Training and Using Disciple Agents: A Case Study in the Military Center of Gravity Analysis Domain. *AI Magazine*, 23(4), 51–68. AAAI Press, Menlo Park, California, 2002.